

LAPORAN PRAKTIKUM STRUKTUR DATA  
STACK PADA JAVA



DOSEN PENGAMPU :

Dr. Wahyudi, M.T.

OLEH :

RIFKI YULIANDRA

NIM.2311532011

DEPARTEMEN INFORMATIKA  
FAKULTAS TEKNOLOGI INFORMASI  
UNIVERSITAS ANDALAS  
PADANG

2024

## *Stack Pada Java*

### I. TUJUAN

- 1.1. Memahami apa yang dimaksud dengan stack
- 1.2. Memahami cara membuat stack
- 1.3. Memahami cara membuat stack dengan menggunakan interface
- 1.4. Memahami cara mencari nilai maksimum dari stack.
- 1.5. Memahami cara konversi postfix dari stack

### II. TEORI

Stack merupakan struktur data fundamental yang mengikuti prinsip Last In First Out (LIFO) atau yang sering disebut dengan "tumpukan". Imagine sebuah tumpukan piring kotor di kantin. Ketika kita selesai menggunakan piring, kita meletakkannya di atas tumpukan. Sebaliknya, ketika kita ingin menggunakan piring bersih, kita ambil piring dari bagian paling atas tumpukan tersebut. Konsep ini lah yang mendasari operasi pada stack. Elemen yang terakhir kali dimasukkan (pushed) ke dalam stack akan menjadi elemen pertama yang dikeluarkan (popped) ketika dibutuhkan.

Stack memiliki banyak sekali aplikasi dalam dunia pemrograman. Salah satu penggunaan yang paling umum adalah untuk membalikkan urutan data. Misalnya, kita dapat menggunakan stack untuk membalikkan urutan kata dalam sebuah kalimat. Stack juga dapat digunakan untuk menyimpan history dari operasi yang dilakukan, sehingga kita dapat melakukan undo atau redo terhadap suatu aksi. Selain itu, stack juga berperan penting dalam implementasi algoritma tertentu, seperti Depth-First Search (DFS) yang biasa digunakan untuk traversal pada graf.

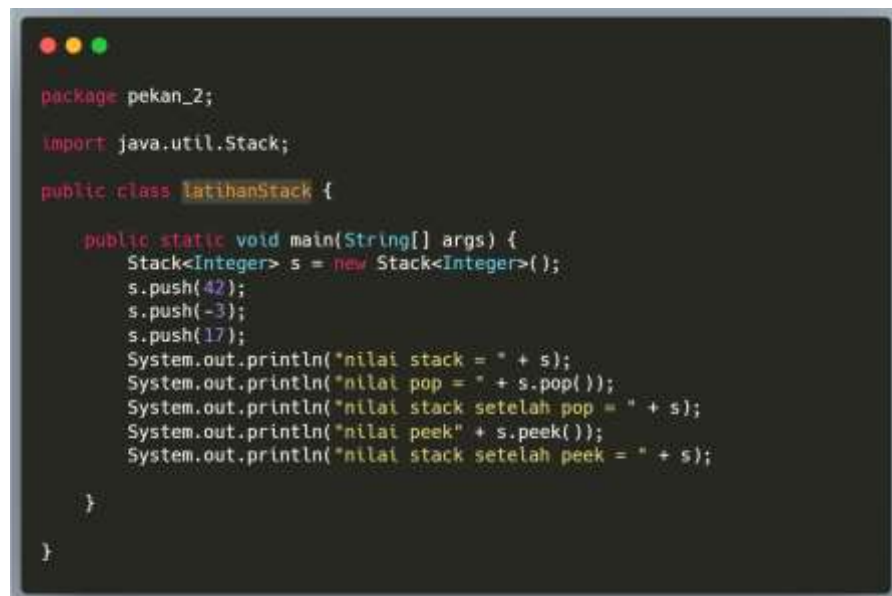
Operasi dasar yang umum dilakukan pada stack adalah push dan pop. Operasi push digunakan untuk memasukkan elemen baru ke dalam stack, sedangkan operasi pop digunakan untuk mengeluarkan elemen teratas dari stack. Selain push dan pop, beberapa implementasi stack juga menyediakan operasi lain seperti peek untuk melihat elemen teratas tanpa

mengeluarkannya, dan `isEmpty` untuk mengecek apakah stack tersebut kosong.

Dalam bahasa pemrograman, stack dapat diimplementasikan menggunakan array atau linked list. Implementasi menggunakan array umumnya lebih sederhana dan efisien untuk operasi akses acak, namun memiliki keterbatasan ukuran yang tetap. Sementara itu, implementasi menggunakan linked list lebih fleksibel dari segi ukuran namun umumnya membutuhkan memory overhead yang lebih besar dibandingkan dengan array. Pemilihan implementasi yang tepat tergantung pada kebutuhan dan karakteristik program yang sedang dikembangkan.

### III. LANGKAH KERJA PRAKTIKUM

#### 3.1. Membuat Stack sederhana



```
package pekan_2;

import java.util.Stack;

public class LatihanStack {

    public static void main(String[] args) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(42);
        s.push(-3);
        s.push(17);
        System.out.println("nilai stack = " + s);
        System.out.println("nilai pop = " + s.pop());
        System.out.println("nilai stack setelah pop = " + s);
        System.out.println("nilai peek" + s.peek());
        System.out.println("nilai stack setelah peek = " + s);
    }
}
```

Deklarasi variabel `s` bertipe `Stack<Integer>` menandakan bahwa variabel `s` akan menyimpan data bilangan bulat (`Integer`). Data bilangan bulat 42, -3, dan 17 dimasukkan ke dalam stack `s` menggunakan method `push()`. Data paling atas dari stack `s` diambil dan dicetak menggunakan method `pop()`. Hasilnya akan dicetak ke layar. Data paling atas dari stack `s` dicetak tetapi tidak dihapus menggunakan method `peek()`. Hasilnya akan dicetak ke layar. Baris kelima dan delapan, method `println()` digunakan

untuk mencetak isi dari stack s sebelum dan sesudah method `pop()` dan `peek()` dipanggil.

Maka output yang dihasilkan:

```
<terminated> latihanStack [Java Application] C:\Users\U
nilai stack = [42, -3, 17]
nilai pop = 17
nilai stack setelah pop = [42, -3]
nilai peek-3
nilai stack setelah peek = [42, -3]
```

### 3.2. Contoh stack yang lain

```
package pekan_2;

public class contohStack {

    public static void main(String[] args) {
        ArrayStack test = new ArrayStack();
        Integer[] a = {4, 8, 15, 16, 23, 42}; // autoboxing allow this
        for (int i = 0; i < a.length; i++) {
            System.out.println("nilai A " + i + " = " + a[i]);
            test.push(a[i]);
        }
        System.out.println("size stacknya: " + test.size());
        System.out.println("paling atas: " + test.top());
        System.out.println("nilainya " + test.pop());
    }
}
```

Deklarasi variabel `test` bertipe `ArrayStack` menandakan bahwa variabel `test` akan menyimpan data menggunakan kelas `ArrayStack`. Baris ketiga, array `a` dideklarasikan dan diinisialisasi dengan beberapa nilai bilangan bulat. Baris lima sampai sepuluh, setiap nilai dari array `a` dimasukkan ke dalam stack `test` menggunakan method `push()`. Setiap nilai yang dimasukkan akan dicetak ke layar. Baris dua belas, method `size()` dipanggil untuk mendapatkan jumlah data yang ada di stack `test` dan dicetak ke layar. Baris tiga belas, method `top()` dipanggil untuk mendapatkan nilai paling atas dari stack `test` dan dicetak ke layar. Baris empat belas, method `pop()` dipanggil untuk mengambil dan mendapatkan nilai paling atas dari stack `test` dan dicetak ke layar.

```

package pekan_2;

public class ArrayStack<E> implements Stack2<E>{
    public static final int CAPACITY = 1000;
    // DEFAULT ARRAY CAPACITY
    private E[] data; // generic array used for storage
    private int t = -1;

    public ArrayStack() {
        this(CAPACITY);
    } // constructs stack with default capacity

    public ArrayStack(int capacity) {
        //construct stack with given capacity
        data = (E[]) new Object[capacity];
    }

    public int size() {
        return (t + 1);
    }

    public boolean isEmpty() {
        return (t == -1);
    }

    public void push(E e) throws IllegalStateException {
        if (size() == data.length) {
            throw new IllegalStateException("Stack is full");
        }
        data[++t] = e; // increment t before storing new item
    }

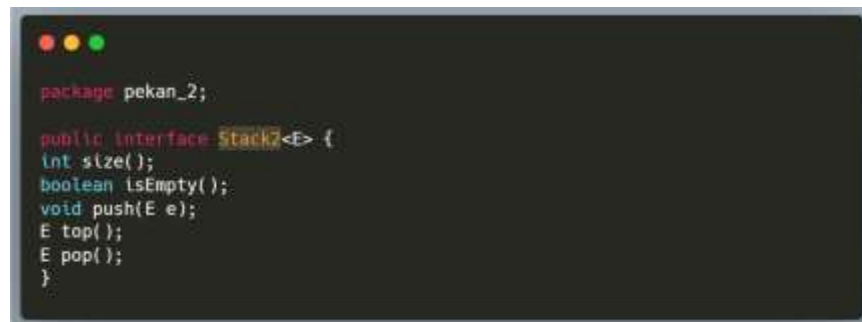
    public E top() {
        if (isEmpty()) {
            return null;
        }
        return data[t];
    }

    public E pop() {
        if (isEmpty()) {
            return null;
        }
        E answer = data[t];
        data[t] = null; // reference to help garbage collection
        t--;
        return answer;
    }
}

```

Program ini mendefinisikan kelas **ArrayStack** yang merupakan implementasi dari struktur data stack menggunakan array. Kelas ini mengimplementasikan interface **Stack2** yang juga didefinisikan oleh pemrogram. Deklarasi variabel **data** bertipe **E[]** menandakan bahwa variabel **data** akan menyimpan data dengan tipe yang sesuai dengan tipe generic **E**. Variabel **t** digunakan sebagai indeks untuk menunjukkan posisi data paling atas di stack. Konstruktor kedua, **ArrayStack(int capacity)**, digunakan untuk menginisialisasi stack dengan kapasitas yang diberikan. Dalam hal ini, kapasitas default adalah 1000. Method **size()** digunakan untuk mendapatkan jumlah data yang ada di stack. Method **isEmpty()** digunakan untuk mengecek apakah stack kosong atau tidak. Method **push(E e)** digunakan untuk menambahkan data ke stack. Jika kapasitas stack

sudah penuh, method akan melemparkan exception **IllegalStateException**. Method **top()** digunakan untuk mendapatkan nilai data paling atas di stack. Jika stack kosong, method akan mengembalikan nilai **null**. Method **pop()** digunakan untuk mengambil dan mendapatkan nilai data paling atas di stack. Jika stack kosong, method akan mengembalikan nilai **null**. Setelah data diambil, posisi data paling atas di stack akan diupdate dengan **null** untuk membantu pemrosesan garbage collection. Program **contohStack** menggunakan kelas **ArrayStack** untuk menyimpan beberapa nilai bilangan bulat dan menggunakan method-method yang tersedia di kelas **ArrayStack** untuk mengelola data tersebut.

A screenshot of a code editor with a dark background and light-colored text. The code defines a package and an interface. The package is named 'pekan\_2'. The interface is named 'Stack2' and is generic with type parameter 'E'. It contains five methods: 'size()' returning 'int', 'isEmpty()' returning 'boolean', 'push(E e)' returning 'void', 'top()' returning 'E', and 'pop()' returning 'E'.

```
package pekan_2;

public interface Stack2<E> {
    int size();
    boolean isEmpty();
    void push(E e);
    E top();
    E pop();
}
```

Program ini mendefinisikan interface **Stack2** yang merupakan spesifikasi dari struktur data stack. Interface ini mendefinisikan beberapa method yang harus diimplementasikan oleh kelas yang mengimplementasikan interface ini. Method `size()` digunakan untuk mendapatkan jumlah data yang ada di stack. Method `isEmpty()` digunakan untuk mengecek apakah stack kosong atau tidak. Method `push(E e)` digunakan untuk menambahkan data ke stack. Method `top()` digunakan untuk mendapatkan nilai data paling atas di stack. Method `pop()` digunakan untuk mengambil dan mendapatkan nilai data paling atas di stack. Kelas **ArrayStack** yang telah dijelaskan sebelumnya mengimplementasikan interface **Stack2** dan menyediakan implementasi untuk semua method yang didefinisikan di interface ini. Program **contohStack** menggunakan kelas **ArrayStack** untuk

menyimpan beberapa nilai bilangan bulat dan menggunakan method-method yang tersedia di kelas **ArrayStack** untuk mengelola data tersebut.

Maka output yang dihasilkan program tersebut adalah

```
<terminated> contohStack [Java]
nilai A 0= 4
nilai A 1= 8
nilai A 2= 15
nilai A 3= 16
nilai A 4= 23
nilai A 5= 42
size stacknya: 6
paling atas: 42
nilainya 42
```

### 3.3. Mencari Nilai Maksimum dari Stack

```
package pekan_2;

import java.util.Stack;

public class NilaiMaksimum {

    public static int max(Stack<Integer> s) {
        Stack<Integer> backup = new Stack<Integer>();
        int maxValue = s.pop();
        backup.push(maxValue);
        while (!s.isEmpty()) {
            int next = s.pop();
            backup.push(next);
            maxValue = Math.max(maxValue, next);
        }
        while (!backup.isEmpty()) {
            s.push(backup.pop());
        }
        return maxValue;
    }


    public static void main(String[] args) {
        Stack<Integer> s = new Stack<Integer>();
        s.push(70);
        s.push(12);
        s.push(20);
        System.out.println("Nilai maksimum " + max(s));
        System.out.println("Stack Teratas " + s.peek());
        System.out.println("isi stack " + s);
    }
}
```

Method `max(Stack<Integer> s)` menggunakan stack **backup** untuk menyimpan data yang diambil dari stack **s** sebelum mencari nilai maksimum. Setelah mencari nilai maksimum, data yang disimpan di stack **backup** akan dikembalikan ke stack **s**.

Method `main(String[] args)` menggunakan kelas **NilaiMaksimum** untuk mencari nilai maksimum dari stack yang diberikan. Setelah mencari nilai maksimum, method akan mencetak nilai maksimum, nilai data paling atas di stack, dan isi stack ke layar. Maka akan didapati output sebagai berikut.

```
<terminated> NilaiMaksimum [Java A
Nilai maksimum 70
Stack Teratas 20
isi stack [70, 12, 20]
```

### 3.4. Melakukan konversi terhadap PostFix



```
package pekan_2;

import java.util.Scanner;
import java.util.Stack;

public class StackPostfix {

    public static int postfixEvaluate (String expression) {
        Stack<Integer> s = new Stack<Integer>();
        Scanner input = new Scanner(expression);
        while (input.hasNext()) {
            if (input.hasNextInt()) {
                s.push(input.nextInt());
            } else { // an operator
                String operator = input.next();
                int operand2 = s.pop();
                int operand1 = s.pop();
                if (operator.equals("+")) {
                    s.push(operand1 + operand2);
                } else if (operator.equals("-")) {
                    s.push(operand1 - operand2);
                } else if (operator.equals("*")) {
                    s.push(operand1 * operand2);
                } else {
                    s.push(operand1 / operand2);
                }
            }
        }
        return s.pop();
    }

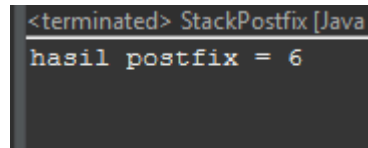
    public static void main(String[] args) {
        System.out.println("hasil postfix = " + postfixEvaluate("5 2 4 * + 7 -"));
    }
}
```

Method `postfixEvaluate(String expression)` menggunakan stack `s` untuk menyimpan data yang diambil dari ekspresi postfix. Setiap kali method menemukan angka, angka tersebut akan dimasukkan ke stack. Setiap kali method menemukan operator, method akan mengambil dua nilai teratas dari stack, melakukan operasi terhadap kedua nilai tersebut, dan mengembalikan hasilnya ke stack.



Method `main(String[] args)` menggunakan kelas **StackPostfix** untuk mengkonversi dan mengevaluasi ekspresi postfix yang diberikan. Setelah mengkonversi dan mengevaluasi ekspresi postfix, method akan mencetak hasilnya ke layar.

Maka akan didapati output sebagai berikut.



```
<terminated> StackPostfix [Java]
hasil postfix = 6
```

#### IV. KESIMPULAN

Program-program yang telah diberikan mengilustrasikan beberapa fitur dari struktur data stack. Program pertama, `latihanStack`, menginisialisasi stack dengan tiga elemen, mengeluarkan satu elemen, dan melakukan operasi peek untuk mengambil elemen teratas tanpa menghapusnya. Program kedua, `contohStack`, menggunakan kelas `ArrayStack` untuk menambahkan elemen ke dalam stack, mencetak elemen teratas, dan mengeluarkan elemen dari stack. Kelas `ArrayStack`, yang merupakan implementasi dari interface `Stack2`, memiliki metode untuk mendapatkan ukuran dan memeriksa apakah stack kosong. Ia juga memiliki metode untuk menambahkan, mengeluarkan, dan mengambil elemen teratas dari stack. Program `contohStack` menggunakan kelas `ArrayStack` ini untuk menambahkan serangkaian angka ke dalam stack dan kemudian mencetak elemen teratas dan mengeluarkan elemen dari stack.

Program `NilaiMaksimum` menggunakan stack untuk menemukan nilai maksimum dalam stack integer yang diberikan. Ia membuat stack cadangan untuk menyimpan elemen asli dan mengeluarkan setiap elemen dari stack input, mengawasi nilai maksimum. Setelah semua elemen dicek, ia memasukkan kembali elemen stack cadangan ke dalam stack input dan mengembalikan nilai maksimum yang ditemukan. Program `StackPostfix` mengevaluasi ekspresi postfix menggunakan stack. Ia membaca setiap token dari ekspresi input, dan jika ia adalah bilangan bulat, dipush ke dalam stack. Jika ia adalah operator, pop dua elemen dari stack, melakukan operasi, dan

push kembali hasilnya ke dalam stack. Kemudian, ia mengembalikan elemen teratas dari stack sebagai hasil evaluasi ekspresi postfix.

Kesimpulannya, program-program Java ini mengilustrasikan beberapa operasi stack, seperti push, pop, peek, dan mencari nilai maksimum dalam stack. Ia juga menunjukkan cara mengevaluasi ekspresi postfix menggunakan stack. Program-program ini dapat berguna untuk memahami dasar-dasar dari struktur data stack dan aplikasinya.